An Organized Repository of Ethereum Smart Contracts' Source Codes and Metrics

Giuseppe Antonio Pierro, Roberto Tonelli, Michele Marchesi

An Organized Repository of Ethereum Smart Contracts Source Code and Metrics

Giuseppe Antonio Pierro*, Roberto Tonelli†, Michele Marchesi†
*Université de Lille, Inria, CNRS, Centrale Lille, UMR 9189 – CRIStAL. Lille, France.
†Dep. of Mathematics and Computer Science. University Of Cagliari. Cagliari, Italy.

Abstract

Many empirical software engineering studies show that there is a great need for repositories where source code is acquired, filtered and classified. During the last few years, Ethereum block explorer services have emerged as a popular project to explore and search Ethereum blockchain data such as transactions, addresses, tokens, smart-contracts' source code, prices and other activities taking place on the Ethereum blockchain. Despite the availability of this kind of services, retrieving specific information useful to empirical software engineering studies, such as the study of smart-contracts' software metrics, might requires many sub-tasks, such as searching specific transactions in a block, parsing files in HTML format and filtering the smart-contracts to remove duplicated code or unused smart-contracts.

In this paper we afford this problem creating *Smart Corpus'*, a Corpus of Smart Contracts in an organized, reasoned, and up to date repository where Solidity source code and other metadata about Ethereum smart contracts can easily and systematically be retrieved. We present the Smart Corpus' design and its initial implementation and we show how the data-set of smart contracts' source code in a variety of programming languages can be queried and processed, to get useful information on smart contracts and their software metrics.

The Smart Corpus aims to create a smart-contracts' repository where smart contracts data (source code, ABI and byte-code) are freely and immediately available and also classified based on the main software metrics identified in the scientific literature. Smart-contracts source code has been validated by EtherScan and each contract comes with its own associated software metrics as computed by the freely available software PASO. Moreover, the Smart Corpus can be easily extended, as the number of new smart-contracts increases day by day.

Index Terms

Ethereum blockchain, Solidity programming language, smart-contracts, software metrics, corpus

I. Introduction

With the advent of blockchain technology as a mainstream technological innovation many researchers and software developers started investigating the new possibilities for software products relying on such infrastructure. Second generation blockchains offer the possibility to code the so called smart-contracts in a Turing complete programming language where all the main operations of traditional software systems can be carried on. The paradigmatic reference is the Ethereum blockchain which offers the possibility to deploy and execute decentralized applications (dApps) which are mainly coded in Solidity, presently the most adopted programming language (1).

Coding smart-contracts which run in a blockchain environment has its peculiarities and constraints and differs from coding in traditional out-of-chain contexts. One of the major differences is the immutability of deployed code, so that if bugs or bad smells are introduced into a smart-contract these cannot be fixed afterwards with patches. Another contract must be deployed in substitution of the former and users must be well advised not to use the wrong code. Another main issue is the interaction with the blockchain by means of transactions where information exchange can occur only between blockchain internal components. Furthermore, memory occupation on blockchain typically has a cost that developers want to reduce and chaining all the blocks poses limits to the reasonable space available for each smart-contract imposing practical constraints to source code size.

This new programming paradigm poses major challenges also to expert developers so that famous failures are commonly found in blockchain software (2; 3). The novelty of the paradigm largely contributes to these faults, since developers do not have historical records or examples where to learn from previous code, as it happens in traditional software coding, where software reuse and coding by imitation are reference practices to help in coding better quality software. Another issue is the lack of reference measures, such as quality, complexity or coupling metrics, which are extensively used in out-of-chain software production to keep software projects under control (4).

The situation is slowly changing for what concerns historical records (even if history is quite recent) of software code, since the Ethereum blockchain can now count on up to 1.5 million deployed smart-contracts, which have been used and running in the last few years. Access to the source code of this body of smart-contracts is still a challenge since the transparency and the open access granted by the public blockchains regards only data registered in the blocks, where only contracts' byte-code is available.

To access the smart-contracts' source code, the developers must resort to other means or to code repositories, such as the classical GitHub or similar resources. Fortunately, in the last few years, EtherScan[1] and other web sites have started providing

[1]https://etherscan.io/

smart-contracts checking as a service, so that Ethereum developers can submit their source code to be analyzed and the source code is made available afterword by the website. However, there is an odd side of the medal for many reasons: the access to this body of knowledge is far from easy and far from fast; it is not structured and organized; the smart-contracts metrics are not available and must be computed separately. All these tasks can and need to be automatized, to save developers' time and work, as well as computational resources. Indeed, in the last few years, a number of research papers have been published reporting findings based on smart-contracts' source code, mined from GitHub or some Ethereum block explorer such as EtherScan (5; 6; 7; 8). However, when conducting this kind of empirical research on smart-contracts with data from Ethereum blockchain, the tasks above mentioned need to be carried out by the developers themselves. The first task is the downloading of the smart-contracts' source code to be analyzed. One way to download smart-contracts' source code data is to inspect open-source software (OSS) project repositories such as GitHub (9).

Another way to perform this task is to use some Ethereum block explorer. These web services allow users to find the desired information by directly accessing the Ethereum blocks, by using a unique identifier or sequentially searching several blocks (10). Some of these Ethereum block explorers provide RESTful Web services, which allow the users to obtain a JSON format payload containing various data. These data may be related to a current or past state of the Ethereum blockchain: an example may be the list of transaction addresses included in a given block of the Ethereum blockchain. This activity might be tedious and time-consuming (11), when conducted by a single user/developer/researcher. Furthermore, the obtained smart-contracts' data-set can consist of duplicated smart contracts, i.e. smart-contracts having different addresses but with the same code.

In this work we tackle these problems and propose an organized, easy to use, large and available software repository for Ethereum smart-contracts source code and metrics where users, researchers, blockchain start-ups and developers can take advantage of the body of knowledge collected during the last few years. This paper thus proposes Smart Corpus, a repository that provides the users with an interface, which allows to search and download smart contracts' source code. The user interface is available at the following online address: https://aphd.github.io/smac-corpus/. The main challenge of the implementation lies in the fact that the Ethereum blockchain stores a massive amount of heterogeneous data, smart-contracts included, which enormously grow in time. For this reason, Smart Corpus was designed to be scalable, by adopting the latest cutting-edge technology, such as document-oriented database, graph query language and serverless computing platform (12).

II. Related Work

A. Previous Literature on Software Corpus Analysis

Gabel and Su (13) built and studied a corpus of open source software written in three of the most widely used languages: C, C++, and Java. The corpus contains six thousand software projects corresponding to 430 million lines of source code. The authors measured the degree to which each project of the corpus can be "assembled" solely from portions of the corpus, thus providing a precise measure of "uniqueness". Their primary contribution is to provide a quantitative answer to the following question: *how unique is software?* Our work also aims to answer this question, because many smart contracts written in the Solidity language have code that is a replication of other smart contracts, although presenting different addresses, as we will see in Section III-B. Our goal is therefore to answer the question: *how unique are smart contracts written in Solidity?* in order to provide a corpus that is composed by smart contracts which can be distinguished from each other.

Tempero and co-authors (14) presented the "Qualitas Corpus", a curated collection of open-source Java systems. The corpus reduced the time needed to find, collect, and organize the necessary source code sets to the time needed to download the corpus. The metadata provided with the corpus explicitly indicates the metrics calculated to identify the main features of the source code: the number of code lines, the number of classes, etc. Our work also aims to present a curated collection of smart-contracts equipped with a set of metadata with the aim of allowing experts in the blockchain field to perform static analysis.

B. Static Analysis on Smart Contract Code

There is a number of scientific publications having as objective the analysis of the smart-contracts' source code, also testifying the scientific community's interest in advancing the knowledge about the characteristics of smart-contracts' code structure.

Hegedus (15) developed a metric calculator for Solidity code, inspired by the work by Tonelli and collaborators (8). The metric calculator uses a parser to generate an abstract syntax tree (AST), on which it computes various software metrics, such as the number of code lines for each smart-contract, the Cyclomatic Complexity, the number of functions, the number of parameters for each function. This command-line tool is written in Java and is available on GitHub without license indication since February 2018 [2]. By using this tool, he calculated and published software metrics results for 10,206 Solidity smart-contracts source code files written in Solidity languages. Our work also aims to calculate a set of metadata on the smart-contracts corpus by using a similar software.

[2]https://github.com/chicxurug/SolMet-Solidity-parser

Pinna and colleagues (16) performed a static analysis on 10,174 smart-contracts, deployed in the Ethereum blockchain. The authors showed that some metrics related to smart contracts, such as the number of transactions and the balances follow power-law distributions. Also, they reported that software code metrics in Solidity have (on average) lower values but higher variance than metrics values in other programming languages for standard softwares. Our work is inspired by their research as Smart Corpus is characterized by (some of) the metrics they defined, as we will explain in Section III-C.

Pierro and Tonelli (17) pointed out that even the most experienced users, as software developers of smart contracts are, need to be helped to analyze smart contracts and write a more reliable and secure code. For this reason, an open-source platform [3], called PASO, was proposed as an aid for experts in smart-contracts' static analysis. Their work focused on Ethereum blockchain and smart-contracts written in Solidity. The platform PASO facilitates the debugging of smart-contracts by providing software metrics commonly used to comply with coding guidelines.

C. Related Projects

Other projects, similar to Smart Corpus, have been previously developed, to online access smart-contracts' code deployed in the Ethereum blockchain platform. The projects present specific features and limitations which are summarized in Table I.

TABLE I: Projects' list, main features and limitations

Project's name	**Home Page**	**REST API URL**	**Limitations**
GitHub	https://github.com/	https://developer.git...	Some repositories have restricted access.
Ethplorer	https://ethplorer.io/	https://api.ethplorer...	Requests are limited to 3000/week.
EtherScan	https://etherscan.io/	https://etherscan...	Smart-contracts' addresses are not immediately available.
EtherChain	https://www.etherch...	https://www.etherch...	Smart-contracts' source code is not available.
BlockScout	https://blockscout.com/	https://blockscout...	Smart-contracts' source code is not available.

1) GitHub: GitHub is the largest collaborative source code hosting site built on top of the Git version control system (18). The availability of a comprehensive API has made GitHub a target for many software engineering and online collaboration research efforts (19). GitHub offers just open-source software to the community. In GitHub, there are many works regarding projects written in different programming languages, such as Java, Python and Solidity, which is by far the most commonly used language to write smart-contracts.

The repository proposed in the paper overcomes the following GitHub's limitations:

- The smart contracts' source code collected in GitHub typically has not a direct reference to smart contracts deployed on the blockchain through an Ethereum address, therefore it is hard to find out whether it has been really tested or used on the blockchain.
- GitHub does not implement a search engine to filter smart-contracts based on particular software metrics, such as the number of modifiers or payables. This is due to the fact that some metrics are specific to the type of language employed to write smart-contracts, i.e. Solidity.
- In GitHub there is no information on smart-contracts' use in a real blockchain scenario, on the number of transactions invoking smart-contracts, or the number of tokens associated with each smart contract.
- GitHub does not provide smart contracts ABI or Opcode.

It is highly probable that the users, especially if they are developers or researchers, want to access smart-contracts' source code choosing the features implemented in Smart Corpus, on the basis of its specific software metrics and its real usage on the blockchain.

2) Ethereum Block Explorers: Ethereum block explorers are platforms that allow the users to explore and search the Ethereum blockchain for transactions, addresses, tokens and other activities taking place on the Ethereum blockchain (20). Unlike GitHub, the Ethereum block explorers allow accessing only Ethereum data used in the Ethereum blockchain and thus smart contracts' real use-cases. To date, in the market, there are different Ethereum block explorers, as for instance:

- **Ethplorer**[4] provides an API to access many Ethereum data, such as the balances for specified token, the description of a specific address but it does not allow to access to the smart contracts' code. The full documentation of the Ethpoler API is available at the following address [5]. The Requests to API are limited to 5 per second, 50/min, 200/hour, 2000/24hours, 3000/week.
- **EtherChain**[6] is an explorer for the Ethereum blockchain. Unlike Ethplorer, it claims to provide smart contract code, even though it actually displays the contract byte-code and the Constructor arguments, for a specific smart contract's address.

[3]https://aphd.github.io/paso/
[4]https://ethplorer.io/
[5]https://github.com/EverexIO/Ethplorer/wiki/Ethplorer-API
[6]https://etherchain.org/

EtherChain provides API just to access the Oracle gas price predictions [7], but not the Ethreum data. If the users want to gather Ethereum data from EtherChain, they need to parser the HTML code.

- **BlockScout**[8] provides an API to access the Ethereum data. It claims to have an API to access only the source code of few verified smart contracts. Anyway the addresses list of the verified smart contracts is not available in BlockScout.
- **EtherScan** allows to explore and search the Ethereum blockchain for smart-contracts. However, when downloading the smart-contracts' source code, the block explorer presents some limitation. First, smart-contracts' data and number are huge (on the Giga scale, based on our estimation), but there is a limited API rate of 100 submissions per day per user to retrieve just a smart contract, making the complete download of the data an impossible endeavour [9]. Second, the EtherScan's API does not provide facilities to obtain a list of the smart-contracts's address, as the existing API calls mainly allow navigation from one block to another. Third, a researcher cannot directly and easily explore the smart-contract's source code, but rather has to first inspect any block in Ethereum and then look for all the transactions that involve an address associated with the smart-contract.

III. Research Methodology

Smart Corpus has been designed to provide the users with a reasoned repository, i.e. a repository which is not just a web space where to collect them but also and mainly a service to help the researchers to filter and analyze the Smart contracts' source code. To this aim, Smart Corpus has been planned to perform four main automatic operations on Smart contracts' source code (data): 1) Data Retrieving, 2) Data Cleaning, 3) Data Modelling, 4) Data Querying. Figure 1 shows the Smart Corpus's pipeline of the operations.

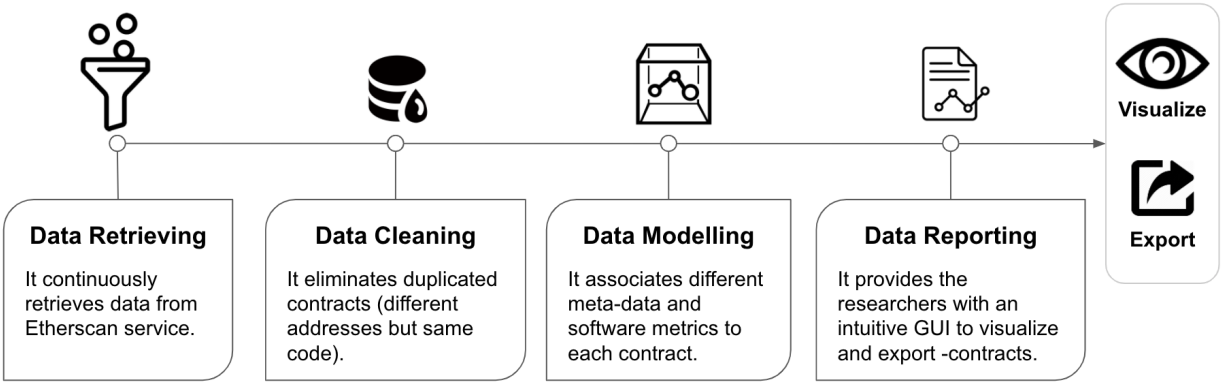


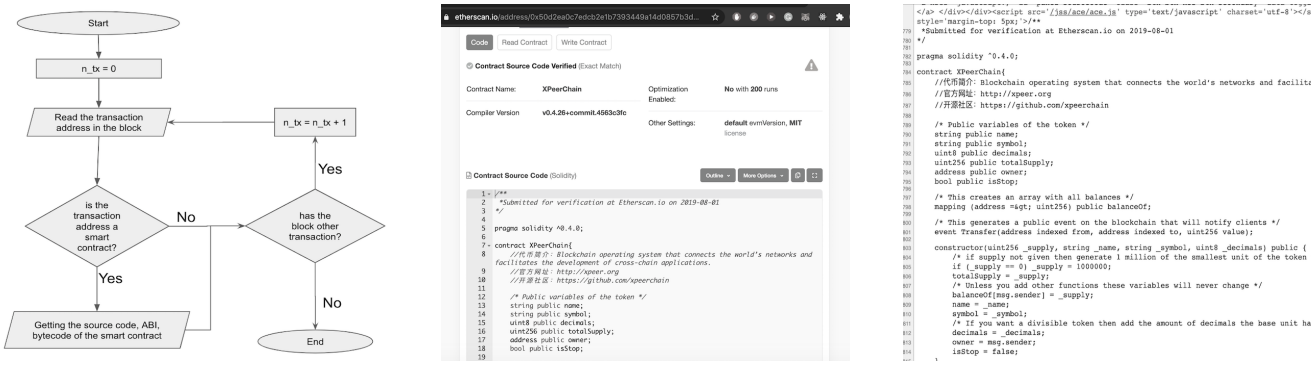
Fig. 1: Smac-Corpus's pipeline model

A. Retrieving Data

We collected smart-contracts' source code, smart-contracts' application binary interface (ABI) and smart-contracts' byte-code through the EtherScan web site, which makes available the source code of a subset of verified smart-contracts deployed on the Etehreum blockchain, though in a laborious way. We instead made this task easier and automatic via a retrieving data script available at the following address [10]. During this phase the blockchain blocks are automatically inspected. Each block is formed by a list of transactions between two different blockchain addresses, which can refer to a wallet or to a smart-contract. The script looks for addresses that refer to a smart-contract and when the source code is available, it downloads the smart-contract's source code, the ABI and the byte-code. The data coming from the source code are not immediately available as they are embedded in the HTML code of the web page provided by EtherScan. Therefore, the script removes the HTML tags and stores the code cleaned up.

Figure 2a shows how Smart Corpus finds the smart-contracts' list in a given block. Figure 2b shows the HTML page where the smart contract code is available. The HTML page containing the smart contract code and the HTML tags is downloaded. Figure 2c shows the HTML code that will be processed to remove the HTML tags and save just the Solidity source code of the smart-contract.

[7]https://www.etherchain.org/api/gasPriceOracle
[8]https://blockscout.com/poa/xdai/
[9]https://etherscan.io/apis#contracts
[10]https://github.com/aphd/solidity-metrics/tree/master/examples



(a) Transactions' list in a block (b) Smart contract's webpage code (c) Smart contract's source code

Fig. 2: Data Retrieving Pipeline. The figures 2a, 2b, 2c shows three different phases to retrieve the smart contracts.

The smart-contracts' code is stored in the filesystem of the Smart Corpus server. Due to the quota limits on queries per second (EtherScan web site allows few connections per second), Smart Corpus contains only a portion of all available smart contracts. However, the retrieving data phase is continuously collecting data, starting from December 10, 2019. To date, thirty thousand of smart-contracts (source code, ABI and byte-code) have been downloaded and made available through Smart Corpus.

B. Cleaning Data

Each smart-contract in the Ethereum blockchain is distinguished from any other smart-contract as it is identified by a unique address, i.e. a hash of 160 bits, and its byte-code is stored on the blockchain (21). Indeed, each time a smart-contract is deployed in the network, either in the main or in the test network, a unique address is associated with the smart-contract even in the case the source code of two or more smart-contracts is the same. However, this is a problem for the analysis of the software metrics, because the smart-contracts are distinguished only on the basis of their address and not of their content. Therefore, Smart Corpus eliminates double contracts in order to provide a clean smart-contracts' corpus, where to perform the analysis. To this aim, duplicate smart-contracts have been defined on the basis of their content, i.e. having the same code despite presenting different address.

C. Modelling Data

Unlike the tools discussed in the related work Section II, Smart Corpus associates different metrics (intrinsic metrics and extrinsic metrics) to the smart-contracts, aiming to facilitate the selection of a smart-contracts' set that meets precise requirements. The metrics associated with the smart-contracts are then stored in a document-oriented database. Figure 3 shows the database schema of a smart contract.

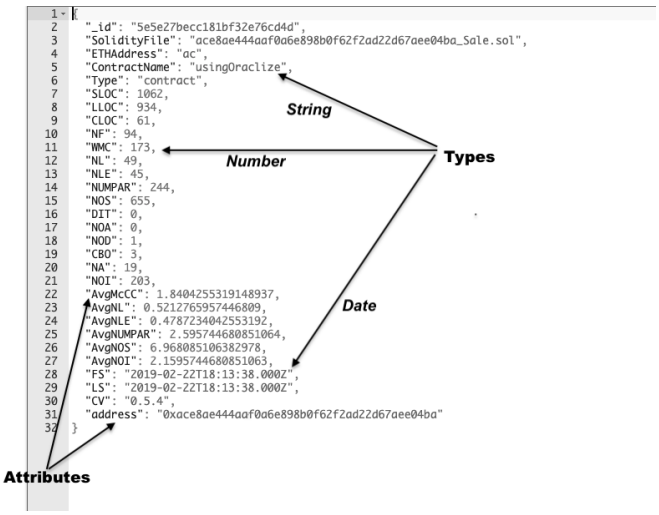


Fig. 3: Smart-Corpus's database schema

1) Smart Contracts' Intrinsic Metrics: The smart contracts' intrinsic metrics are smart-contracts' software metrics which depend on internal properties of the smart contracts' code, such as the number of lines of code, modifiers, payable, etc. Table II shows the smart-contracts' intrinsic software metrics.

TABLE II: Smart-contracts' intrinsic metrics

Name	Description
Pragma	"Pragma" indicates version indicates which version of Solidity compiler is used to prevent issues with future compiler versions.
SLOC	"SLOC" indicates the number of lines in a smart-contracts' source code.
Modifiers	"Modifiers" indicates the number of function modifiers in a smart-contract.
Payable	"Payable" indicates the number of payable functions in a smart-contract.
Mapping	"Mapping" indicates the number of variables of mapping types in a smart-contract.
Address	"Address" indicates the number of variables of address types in a smart-contract.

2) Smart Contract Extrinsic Metric: The smart-contracts' extrinsic metrics are properties depending on external factors rather than the code itself, such as the number of transactions executed to the smart-contracts or the number of tokens associated with the smart contracts. Table III shows the smart-contracts' extrinsic metrics.

TABLE III: Smart-contracts' extrinsic metrics

Name	Description
Transactions	"Transactions" represents the total number of transactions generated by the smart contract (sent or received).
Balance	"Balance" is the amount of crypto coins associated with a smart-contract address.
EtherValue	"EtherValue" is the dollar value associated with a smart-contract address.
Token	"Token" is the value for each token associated with a smart-contract address.
Last_seen	"Last_seen" is the timestamp of the last time the smart-contract was used (sent or received).
First_seen	"First_seen" is the timestamp of the first time the smart-contract was used (sent or received).

D. Filtered Data

The smart contracts' source code is stored in a file system and is organized in folders and subfolders to ease the navigation. Figure 4 shows the sub-directory structure. The first leaf corresponds to the first two letters of the smart contract address and then each directory contains the file having as name the full address of the smart contract and three different extensions, respectively .sol for the Solidity source code, .abi for the ABI, and .bytecode for the byte code.

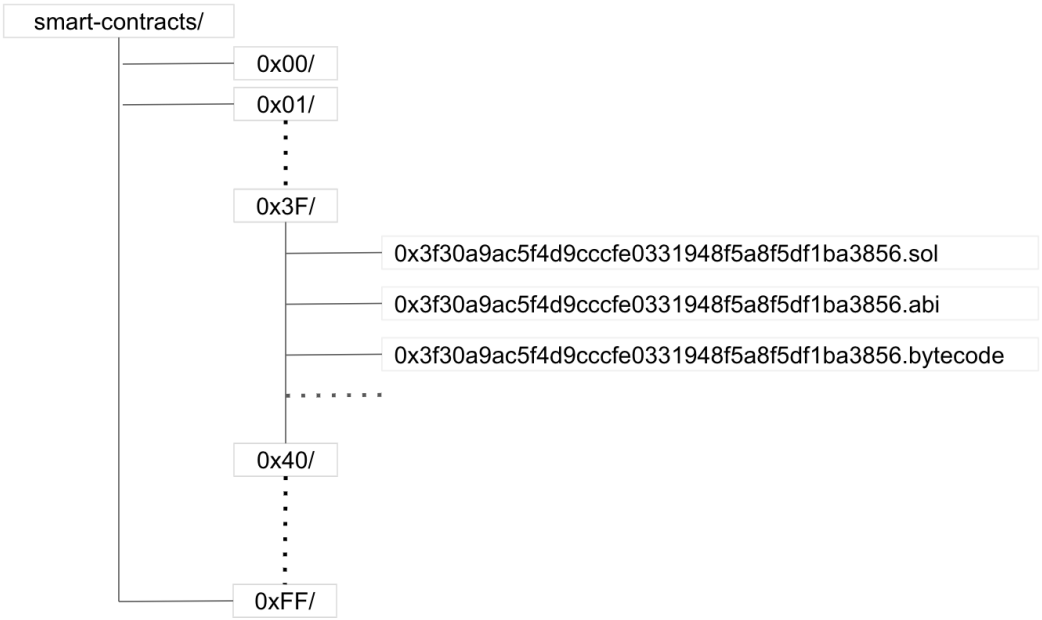


Fig. 4: Smart-contracts' directory structure.

The metadata (both the intrinsic and extrinsic metrics) are stored in a document-oriented database: MongoDB (22). The choice to use a Document-oriented database instead of a relational database such as Mysql is based on:

- Relational databases are prone to deterioration when data-sets overcome a size threshold, while a document-oriented database such as MongoDB comes with inbuilt load balancer, which makes it a better solution in applications with high data load (23). We update MongoDB each day to generate the data archive.
- Unlike relational databases where data is stored in rows and columns, document oriented databases store data in documents. The documents typically use a structure similar to JSON (JavaScript Object Notation), they indeed provide a natural way to model data that is closely aligned with object oriented programming. Each document is considered as an object in object oriented programming, similarly each document is a JSON in document-oriented database. The concept of schema in document databases is dynamic: every document might contain a different number of fields. This is useful when modeling unstructured and polymorphic data. Also, document databases allow robust queries: any combination of fields in the document can be combined for querying data (24).

E. User Interface

The Smart Corpus' graphical user interface (GUI) allows users to access the smart-contracts' repository. There are two way to access the smart-contracts' repository: through the "HTML user interface" and through a "GraphiQL application", both of them via a Web browser.

1) Smart Corpus HTML user interface: The Smart Corpus HTML user interface is publicly available since January 2020 [11]. Figure 6 shows the different components of the GUI.

- At the top, the user can find the form where to filter the smart-contracts. The form is made of a number of drop-down lists, each one corresponding to a different metric and a submit button to perform the research. The GUI form allows the user to inspect smart contracts based on some metadata, such as the "pragma version", and software metrics, such as the numbers of "modifiers" and/or the numbers of "payable".
- Below the form, the smart contracts filtered by the user are displayed. For readability reasons, only a part of the smart-contracts metrics are presented in the table layout format. Each column header in the table indicates the name of a metric associated to smart-contracts. While the HTML-GUI displays just some metrics, the user can access to all the metrics and to the smart-contracts' source code by selecting the checkbox displayed on the right of the smart-contract address and clicking on the red button "download". The user can also access the original repository where the smart-contract was retrieved, i.e. the EtherScan service.

2) Smart Corpus GraphQL application: Graph Query Language (GQL) is a full data query language to implement web-based services, centered on high-level abstractions, such as schemas, types, queries, and mutations. GQL is a domain-specific language internally developed in Facebook from 2012 onward and publicly announced in 2015, with the release of a draft language specification. The language was conceived with the following goals:

- To reduce possible overload of data transfer relative to REST-like web service models, in terms of both the amount of data unnecessarily transferred, and the number of separate queries required to do it.
- To reduce the potential of errors caused by invalid queries on the part of the client. In particular, with the GQL application, the user can execute "type introspection", i.e. the user can examine the type or properties of an object at runtime. For example, thanks to the introspection queries the user can find out both the intrinsic and the extrinsic metrics associated with a specific smart-contract while typing the query.

Figure 5 shows an example query and its result.

Fig. 5: Example with the use of variables to filter a query result with GraphQL.

[11] https://aphd.github.io/smac-corpus/

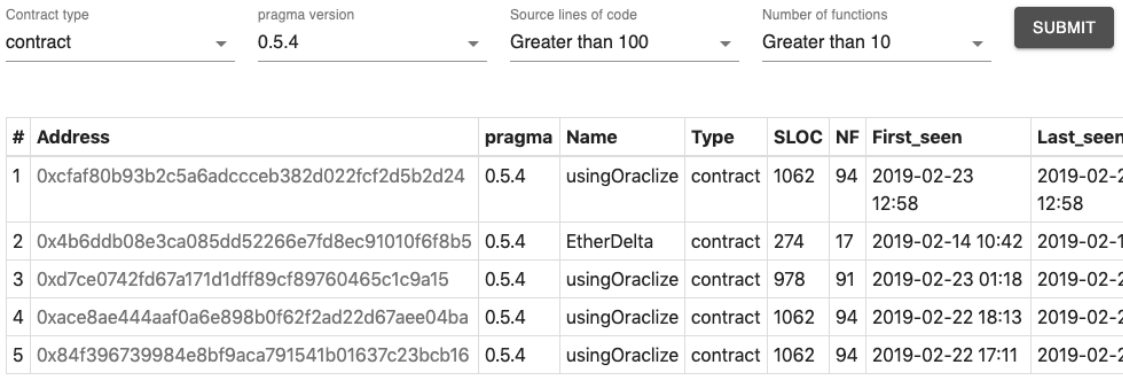
Smart Corpus GQL application, unlike the Smart Corpus HTML user interface is still in development and testing phase. However, the Smart Corpus GQL application source code is publicly available and can be downloaded and deployed on any platform having the software requirements specified in its documentation [12].

F. Use Case

A use case for Smart Corpus might concern a researcher interested in the static analysis of smart contracts. For example, the researcher might be interested in performing an analysis of smart-contracts written with a particular version of the Solidity language, 6.0, and having at least a payable function in the smart-contract. The research of smart contracts that meets these requirements would be very expensive in terms of time, work and computational resources, using a service like EtherScan. Instead, thanks to the Smart Corpus, the user needs to perform only a few steps, as described below:

- connect to the service through the link: https://aphd.github.io/smac-corpus/
- select the option ”version 6.0” from the drop-down menu entitled ”pragma version”.
- select the option ”greater than zero” from the drop-down menu entitled ”number of payables”.
- submit the form by clicking on the button ”submit”.

After few seconds, depending on the number of smart-contracts that meet the requirements specified by the user, the smart-contracts' addresses and the metrics values will be displayed in a table layout format ready to be downloaded.



#	Address	pragma	Name	Type	SLOC	NF	First_seen	Last_seen
1	0xcfaf80b93b2c5a6adccceb382d022fcf2d5b2d24	0.5.4	usingOraclize	contract	1062	94	2019-02-23 12:58	2019-02-23 12:58
2	0x4b6ddb08e3ca085dd52266e7fd8ec91010f6f8b5	0.5.4	EtherDelta	contract	274	17	2019-02-14 10:42	2019-02-14 10:42
3	0xd7ce0742fd67a171d1dff89cf89760465c1c9a15	0.5.4	usingOraclize	contract	978	91	2019-02-23 01:18	2019-02-23 01:18
4	0xace8ae444aaf0a6e898b0f62f2ad22d67aee04ba	0.5.4	usingOraclize	contract	1062	94	2019-02-22 18:13	2019-02-22 18:13
5	0x84f396739984e8bf9aca791541b01637c23bcb16	0.5.4	usingOraclize	contract	1062	94	2019-02-22 17:11	2019-02-22 17:11

Fig. 6: Smart Corpus's User Interface

IV. Results

The Smart Corpus has been in use for 10 months, since December 2019 and 100K smart contracts have been downloaded via the user interface. Until the paper was written (October 2020), Smart Corpus is a curated corpus of 30K smart-contracts's source codes, ABI and byte-codes with related metadata and software metrics. As the time passes, the Smart Corpus is continuously increasing at a rate of 100 smart contracts per day. Figure 7 shows the number of smart-contracts' source codes, ABI and byte-codes retrieved per day since the Smart Corpus was deployed for the first time. For each smart contract Smart Corpus computed extrinsic and intrinsic metrics, as described in Sections III-C2 and III-C1.

[12]https://github.com/aphd/smac-corpus-api

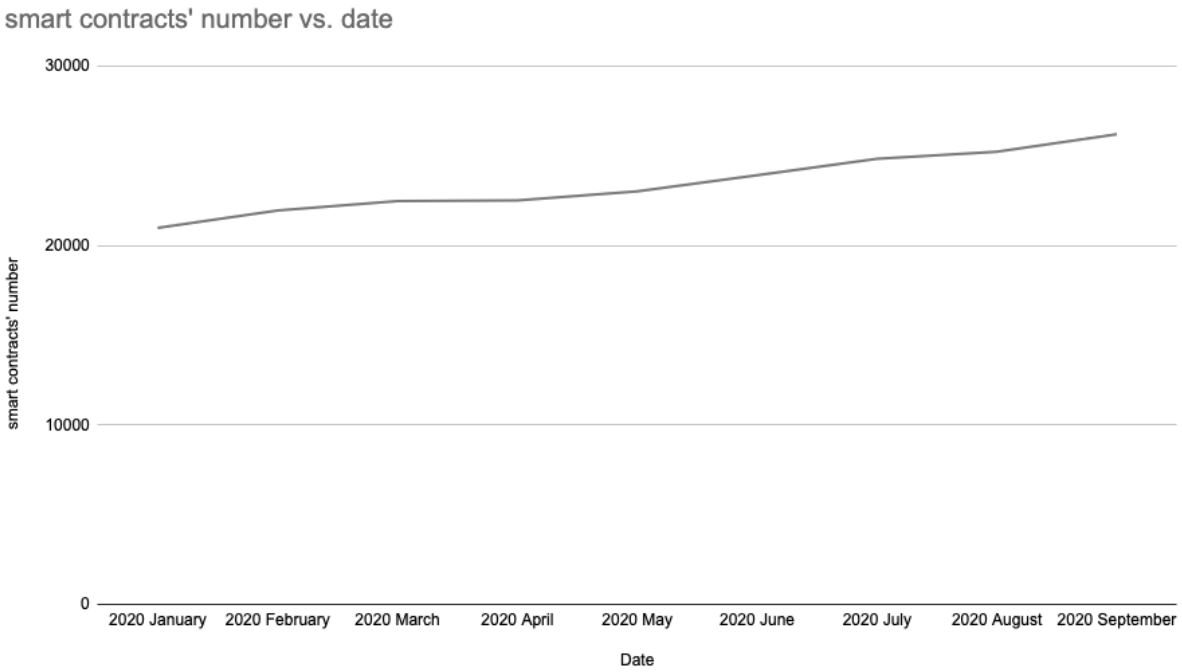


Fig. 7: Number of smart-contracts' collected in Smart Corpus.

Summing up, the Smart Corpus has two GUIs to access data, the HTML GUI and the GraphQL interface. The HTML GUI is described in Section III-E1, while the GraphQL interface is described in Section III-E2. The GraphQL interface gives blockchain researchers the ability to request for exactly what they need. The user can directly access the results via GraphQL interface, as shown in Figure 5.

Unlike the existing repositories (see Section II-C2) which make available the source code in a laborious way, Smart Corpus instead made this task easier and faster. Indeed, one of the advantages of using Smart Corpus lies in the fact that it can reduce the costs in performing the smart contract static analysis. For example, it can be used to easily analyze design and programming patterns for the smart contract programming language.

Even though Smart-Corpus service has been working for a few months and has not been advertised yet, it has already collected 30K smart-contracts, thus providing an interesting and helpful future venue for researchers and software developers interested in the Blockchain. Moreover, Smart Corpus allows to analyze how the industry companies use the Solidity programming language to solve concrete problems in different application areas, such as Healthcare, Insurance, Transportation, Government, Entertainment and Energy.

V. Conclusions and Future Works

In this paper, we described the Smart Corpus project, an effort to bring smart contracts data (source codes, ABIs and byte-codes) to the hands of the research community, providing help for reproducible research and a less time-consuming way to gather data and perform static analysis. The project has already stored several megabytes of data, which correspond to about thirty thousand of smart contracts. This work corresponds to 10 months of data retrieving that are made available to the blockchain scientific community and the blockchain developers in a few seconds. The Smart Corpus data-set has strong potential to provide an interesting venue for research in many software engineering areas including, but not limited to, the best practices for Solidity software development, distributed collaboration, and code paternity and attribution. The Smart Corpus project is in its initial stage of development, but it can already provide useful insight for researchers on smart contracts' coding and everyday use in the blockchain. The corpus will continue to be expanded in content and in the provision of intrinsic and extrinsic metrics, this becoming more and more representative of the Solidity code actually used in the blockchain community.

Appendix

Listing 1 displays a GQL query that returns the smart-contracts's address having more than 20 methods defined in a contract. Listing 2 displays the query results in the JSON format. The query output, in addition to the smart-contract's address, contains various information (intrinsic metrics) such as the number of events, the number of functions, the number of modifiers and the number of payables, as specified by the query 1,

Listing 1: A GQL query for displaying intrinsic metrics.

```
{
    metrics(query:{functions_gt: 20}) {
      adress
      events
      functions
      modifiers
      payable
    }
  }
```

Listing 2: A GQL result displaying intrinsic metrics.

```
{
    "data": {
      "metrics": [
        {
          "contractAddress": "0xb7f4c286851cbf0cbf2fe8ebf40412b196c0e8ad",
          "events": 7,
          "functions": 27,
          "modifiers": 1,
          "payable": 1
        },
        {
          "contractAddress": "0x755cebe8cc53c7cb1e1bb641026a17d37d4aea91",
          "events": 4,
          "functions": 31,
          "modifiers": 1,
          "payable": 4
        },
        {
          "contractAddress": "0xb92aa4a864daf0d6a509e73a9364feba44384965",
          "events": 3,
          "functions": 24,
          "modifiers": 1,
          "payable": 1
        },

        ...

    }
}
```

Listing 3 displays a GQL query that returns some extrinsic metrics of a specific smart-contracts's address. Listing 4 displays the query results in the JSON format. The query output, in addition to the smart-contract's address, contains information such as the total number of transactions generated by the smart contract, the amount of crypto coins associated with the smart-contract's address specified by the query 3,

Listing 3: A GQL query for displaying exstrinsic metrics.

```
{
    metrics(query:{address_eq: "0x536c7efeebff067a69393133b1c87a163a6b0598"}) {
      adress
      transactions
      balance
    }
  }
```

Listing 4: A GQL result displaying exstrinsic metrics.

```
{
    "data": {
      "metrics": [
        {
          "contractAddress": "0x536c7efeebff067a69393133b1c87a163a6b0598",
          "transactions": 639 ,
          "balance": 0 Ether
        }
      ]
    }
}
```


REFERENCES

[1] Peter O'Donovan and Dominic T. J. O'Sullivan. A systematic analysis of real-world energy blockchain initiatives. *Future Internet*, 11(8):174, Aug 2019.

[2] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, Apr 2020.

[3] B. Shala, U. Trick, A. Lehmann, B. Ghita, and S. Shiaeles. Blockchain and trust for secure, end-user-based and decentralized iot service provision. *IEEE Access*, 8:119961–119979, 2020.

[4] Simona Ibba, Andrea Pinna, Maria Lunesu, Michele Marchesi, and Roberto Tonelli. Initial coin offerings and agile practices. *Future Internet*, 10(11):103, Oct 2018.

[5] Alexander Mense and Markus Flatscher. Security vulnerabilities in ethereum smart contracts. In *Proceedings of the 20th International Conference on Information Integration and Web-Based Applications*, page 375–380, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Sidney Amani, Myriam Bégel, Maksym Bortin, and Mark Staples. Towards verifying ethereum smart contract bytecode in isabelle/hol. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, page 66–77, New York, NY, USA, 2018. Association for Computing Machinery.

[7] Hien Tran, Tarek Menouer, Patrice Darmon, Abdoulaye Doucoure, and François Binder. Smart contracts search engine in blockchain. In *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, ICFNDS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Roberto Tonelli, Giuseppe Destefanis, Michele Marchesi, and Marco Ortu. Smart contracts software metrics: a first study, 2018.

[9] Letizia Jaccheri and Thomas Osterlie. Open source software: A source of possibilities for software engineering education and empirical software engineering. In *Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development*, FLOSS '07, page 5, USA, 2007. IEEE Computer Society.

[10] Santiago Bragagnolo, Henrique Rocha, Marcus Denker, and Stéphane Ducasse. Ethereum query language. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, WETSEB '18, page 1–8, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Ying Zhou and Joseph Davis. Open source software reliability model: An empirical approach. In *Proceedings of the Fifth Workshop on Open Source Software Engineering*, 5-WOSSE, page 1–6, New York, NY, USA, 2005. Association for Computing Machinery.

[12] Nane Kratzke. Volunteer down: How covid-19 created the largest idling supercomputer on earth. *Future Internet*, 12(6):98, Jun 2020.

[13] Mark Gabel and Zhendong Su. A study of the uniqueness of source code. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, page 147–156, New York, NY, USA, 2010. Association for Computing Machinery.

[14] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. The qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference*, pages 336–345, 2010.

[15] P. Hegedus. Towards analyzing the complexity landscape of solidity based ethereum smart contracts. In *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 35–39, 2018.

[16] Andrea Pinna, Simona Ibba, Gavina Baralla, Roberto Tonelli, and Michele Marchesi. A massive analysis of ethereum smart contracts empirical study and code metrics. *IEEE Access*, 7:78194–78213, 2019.

[17] Giuseppe Antonio Pierro and Roberto Tonelli. Paso. In *Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, 2020.

[18] Jon Loeliger. *Version control with Git*. O'Reilly Media, Sebastopol, Calif, 2012.

[19] Georgios Gousios and Diomidis Spinellis. Mining software engineering data from github. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, page 501–502. IEEE Press, 2017.

[20] Stefano Bistarelli, Gianmarco Mazzante, Matteo Micheletti, Leonardo Mostarda, and Francesco Tiezzi. Analysis of ethereum smart contracts and opcodes. In Leonard Barolli, Makoto Takizawa, Fatos Xhafa, and Tomoya Enokido, editors, *Advanced Information Networking and Applications*, pages 546–558, Cham, 2020. Springer International Publishing.

[21] Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object oriented design. In *Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA '91, page 197–211, New York, NY, USA, 1991. Association for Computing Machinery.

[22] Kristina Chodorow. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2013.

[23] Miguel Diogo, Bruno Cabral, and Jorge Bernardino. Consistency models of nosql databases. *Future Internet*, 11(2):43, Feb 2019.

[24] Sedick Baker Effendi, Brink van der Merwe, and Wolf-Tilo Balke. Suitability of graph database technology for the analysis of spatio-temporal data. *Future Internet*, 12(5):78, Apr 2020.